Concurrencia: posibilidad de realizar múltiples tareas en simultaneo, No secuencial.

No determinismo : cuando se ejecuta un programa con X datos como entrada, la Resolución final puede variar, ya que la ejecución de los programas no es secuencial, sino que se compite por el procesador. Esto complejiza el DEBUG.

Objetivos de programación concurrente:

1. Ajustar el modelo: de arquitectura de hardware y software al problema del mundo real.
2. Incrementar el rendimiento: mejorando tiempos de respuesta de los sistemas de computo, a través de un enfoque diferente de la arquitectura física y lógica.

Procesos: Un programa concurrente esta compuesto por **procesos**. Cada proceso posee un único flujo de control individual que ejecuta una instrucción y al terminar ejecuta la siguiente instrucción (de manera secuencial).
Un proceso realiza tareas de manera :

- Cooperativa: cada proceso genera datos que otro similar los utilizan para resolver tareas mas complejas
- Competencia: entre ellos para usar un recurso compartido.

Nota: todos los procesos pueden tener estas dos maneras de actuar durante la ejecución de un programa. No tiene por que ser uno solo

En la **competencia** Puede surgir dos grandes problemáticas:

- Deadlock : Esto sucede cuando dos procesos tienen un recurso bloqueado que otro proceso necesita para continuar. Ej : 2 personas tejiendo, una color blanco y otra gris. La 1era necesita el gris y se queda esperándolo pero no devuelve el blanco, la 2da quiere el blanco y mientras espera que este disponible retiene el gris.
- Inanición : Esto sucede cuando en la competencia por un recurso un proceso nunca gane. Supongamos 6 personas tejiendo y un proceso “X” necesita el color blanco, lo espera. Cuando el color se libera, otro proceso que también necesitaba el blanco gana la competencia y ocupa el color. Si esto sucede reiteradas veces y el proceso “X” nunca tuvo el color, se lo denomina inanición.

En la cooperación se divide un problema en pequeñas partes de manera que distintos procesos las resuelvan por separada. Luego se unen estas resoluciones para resolver el problema original.
Ej: las personas tejiendo un pullover, donde cada una hace una parte del mismo (mangas, cuerpo, cuello, etc) y luego los unen.
Es to genera ciertos problemas ya que quien asigna que tipo de tarea resuelve cada parte es compleja.

Procesamiento Secuencial, concurrente y paralelo:

- **Secuencial**: Es cuando se realiza una tarea y al terminarla se pasa a ala siguiente. Es la manera mas sencilla de resolver un problema complejo, ya que nos enfocamos unicamente en una única porción del problema. Esto nos obliga a tener una referencia TEMPORAL, ya que hasta que no se termina lo primero, no se pasa a lo segundo.

- **Paralelo**: Varias acciones ejecutándose en simultaneo, sin compartir recursos (por ende no hay competencia). Este tipo de procesamiento se basa en dividir un problema grande en trozos mas chicos, que se puedan resolver en simultaneo. Esto es mucho mas rápido que el procesamiento secuencial, pero mas costoso en recursos(por la necesidad de no compartir recursos).
  La problemática que surge con este tipo de modelo esta en la distribución de trabajo, la necesidad de comunicarse entre procesos, tratamiento de fallas, necesidad de esperarse en puntos claves
- **Concurrente**: Existen 2. Con paralelismo de hardware y sin.
  - Sin paralelismo: En este caso cita el ejemplo de una maquina que realiza piezas mecánicas, la maquina en un comienzo hace X cantidad de piezas iguales, y luego sigue con la misma proporción pero de otra pieza.
  - Con paralelismo : + de 1 maquina, que pueden estar trabajando en la misma etapa, o diferidas, esto mejora el tema de mejora en tiempos evitando tiempos ociosos.

Paralelismo => Se asocia con la ejecución concurrentemente en múltiples procesadores con el objetivo principal de reducir el tiempo de ejecución.
Concurrencia => Concepto de software no restringido a una arquitectura en particular de hardware ni a un numero determinado de procesadores. Donde 2 o mas programas secuenciales pueden ejecutarse de manera concurrente en el tiempo como tareas o procesos.
Un programa puede tener N procesos habilitados para ejecutarse concurrentemente, y un sistema concurrente puede tener M procesadores donde se puedan ejecutar 1 o mas procesos.

Context Switch : se basa en suspender el proceso actual y se restaura otro, para ello se debe:
1. Salvar el estado actual de la memoria. Agregar el proceso al final de la cola de ready o una cola de wait
2. Sacar un proceso de la cabeza de la cola ready, restaurar su estado y ejecutarlo.

Estado de los procesos:

Procesos: cada proceso tiene su propio espacio de direcciones y recursos
Procesos Livianos (threads/hilos): tienen sus propio contador del programa y su propia pila de ejecución, pero no controla el contexto pesado (ej: tabla de paginas).
Todos los hilos comparten el mismo espacio de direcciones y recurso (los del proceso)
El programador o lenguaje se debe encargar de evitar las interferencias (ej:dos hilos queriendo escribir un mismo dato en simultaneo).
La concurrencia puede ser provista por un lenguaje (java) o SO (unix)

PARTE II
Comunicación entre procesos concurrentes:
1. **Por memoria compartida**: Se basa en el hecho que se guarda información en posiciones de memoria que pueden ser accedidas por cualquier proceso que tenga acceso a la misma. Esto no solo significa memoria física compartida, sino mismo espacio de direcciones. Para ello es necesarios el **BLOQUEAR Y LIBERAR** el acceso a la memoria (para ello usaremos un especie de “semáforo” que nos habilite o no el acceso). Es necesario que la arquitectura de la maquina a usar nos permita utilizar este tipo de método.
2. **Por pasaje de mensajes**: Las unidades de procesamiento se encuentran en una arquitectura distribuida (separadas en maquinas diferentes, sin memoria compartida),

y la comunicación se realizara por medio de mensajes. Para ello es requerido **Canales** lógicos o físicos por donde viajaran los mensajes. Estos mensajes serán sentencias primitivas que nos posibiliten la comunicación.
También puede darse dentro de una misma maquina con memoria compartida, pero los procesos no pueden acceder al espacio de direcciones de otro proceso.

Sincronización entre procesos concurrentes: Es la posesión de información acerca de otro proceso para coordinar actividades. Los procesos se sincronizan:

1. Por **exclusión mutua**: Asegurar que un único proceso tenga acceso a un recurso compartido en un instante de tiempo. Si el programa tiene **secciones críticas** que pueden compartir más de un proceso, la exclusión mutua evita que dos o más procesos puedan encontrarse en la misma sección crítica al mismo tiempo.
   Ej: el uso de una impresora, un periférico, etc.

2. Por **condición**: Permite bloquear la ejecución de un proceso hasta que se cumpla una condición dada.

Interferencia : cuando un proceso toma una acción que invalida las suposiciones hechas por otro proceso. Ej: Dos procesos que usan una misma variable(compartida), de las cuales una modifica el valor de la variable y la otra lo usa para realizar una división, si bien uno puede chequear con un IF si la variable es <>0, la modificación de esta puede ser posterior al chequeo y anterior a la división, provocando que esta ecuación de error y por consiguiente rompe el programa.

Prioridad: un proceso que tiene mayor prioridad puede causar la suspensión (preemption) de otro proceso concurrente, obligándolo a retirarse de la ejecución en un instante dado.
En la materia, este tipo de situaciones no pasan. Esto lo maneja el Sistema operativo.

Granularidad: de una aplicación esta dada por la relación entre el computo y la comunicación.

- **Grano fino**: poco código computable y mucha comunicación. Ej: procesos de muchos hilos.
- **Grano grueso**: mucho código computable y poca comunicación. Ej: CLÚSTER

Nota: Para determinar que tipo de grano debe usarse nos debemos situar en la maquina en la cual se ejecutara nuestra aplicación, dado que una maquina con gran poder de computo pero escaso en comunicación, la opción de grano fino, no seria lo recomendable.

Administración de recursos compartidos : Esto incluye la asignación de recursos compartidos, métodos de acceso a los recursos, bloqueo y liberación e recursos, seguridad y consistencia
Fairness : Es el equilibrio en el acceso a recursos compartidos por todos los procesos.

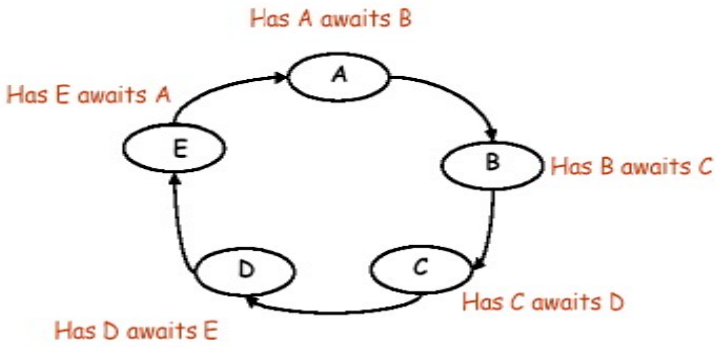
Situaciones indeseables en programación concurrente:

- **Inanición:** un proceso no logra acceder a los recursos compartidos.
- **Overloading** : La carga asignada a un proceso excede su capacidad de procesamiento.

Deadlock,si por error de programación 2 procesos se quedan esperando que el otro libere un recurso compartido. La ausencia de deadlock es una propiedad necesaria en los procesos concurrentes. El deadlock se da debido a 4 propiedades :

1. **Recursos reusables serialmente**: los procesos comparten recursos que pueden usar con exclusión mutua.

2. **Adquisición incremental**: los procesos mantienen los recursos que poseen mientras esperar adquirir recursos adicionales. Cada proceso espera un recurso compartido libre y lo bloquea. Una vez que tiene todos, realiza su función.
3. **No-preemption**: una vez que son adquiridos por un proceso, los recursos no pueden quitarse de manera forzada sino que sólo son liberados voluntariamente.
4. **Espera cíclica**: existe una cadena circular (ciclo) de procesos tal que cada uno tiene un recurso que su sucesor en el ciclo está esperando adquirir.



PARTE III

Concurrencia a nivel hardware.

*- Maquinas C/mono procesador no se puede mejorar mas y de ahí surge el multiprocesador. Generando clúster de multicores (varias maquinas multicores unidas por medio de una red informática). Esto mejora el rendimiento en sistemas que requieren mucho procesamiento (laboratorios, registros etc).

*- Paradigma Híbrido : Surge de poder mezclar comunicación por envío de mensajes (entre distintas maquinas dentro de mismo clúster de equipos), con comunicación por medio de variables compartidas (compariendo info entre procesos dentro de un mismo multicore, hilos de esta maquina) en un mismo programa.

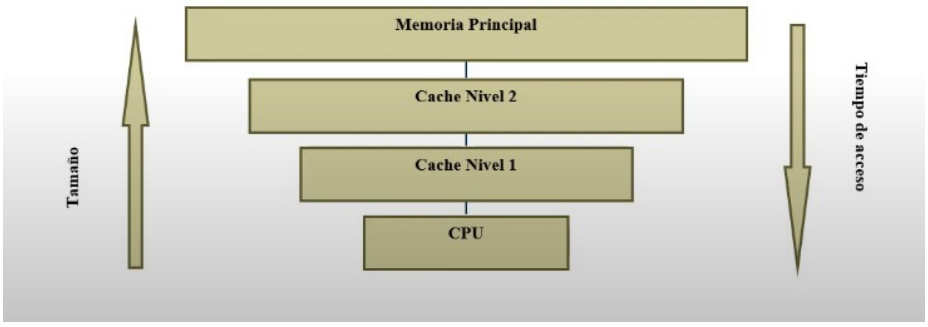
*- Surge como problema:

- Consumo de energía : muchos equipos trabajando entre ellos genera mucho gasto energético (debido al consumo para generar computo) y desprenden mucho calor, el cual se intenta aplacar usando aires acondicionados donde están las maquinas funcionando.

  Este parámetro (el consumo energético) es tomado hoy en día como métrica para verificar que el programa ,ademas de efectivo , consuma lo menos posible.

La forma de aprovechar el paralelismo de maquinas es usando correctamente la programación concurrente y paralela.

Jerarquía de memoria



Cuanto mas cercano a la cpu la memoria es mas chica y mas rápida que en niveles superiores.

Cuanto mas procesos abiertos ejecutándose de manera concurrente, mas cuidado hay que tener con la utilización de la memoria cache, ya que, cuando el programa cambia entre procesos se genera el context swich de datos lo cual es muy costoso a nivel rendimiento. La info que no este cargada en las cache hay que ir a a buscar a disco, por lo tanto se tarda mucho tiempo en realizar estos cambios.

Esto se da debido a que por ejemplo si un procesador esta ejecutando 4 procesos en simultaneo se les otorga ¼ parte de memoria cache a cada proceso, reduciendo el volumen de paginas que cada proceso puede tener cargada en memoria, y por ende aumenta la posibilidad de fallo de paginas.

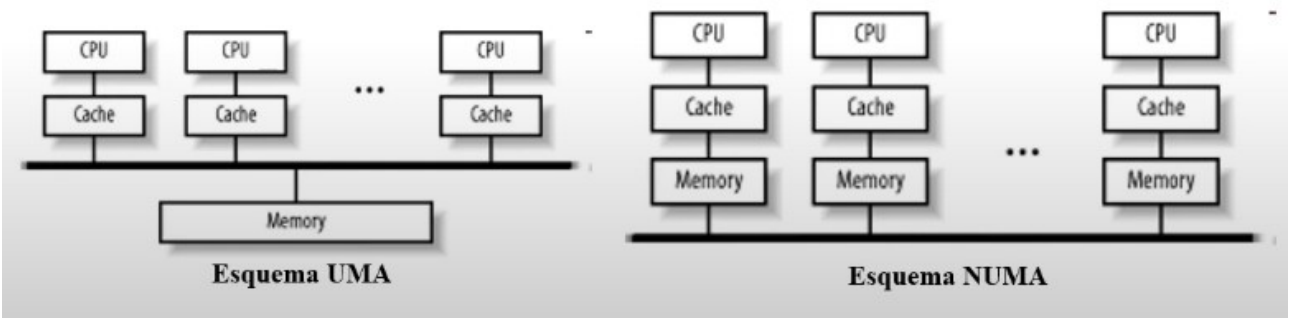
La "**localidad espacial y temporal de datos**" es un concepto utilizado para describir la eficiencia de acceso a los datos en un sistema de almacenamiento.

- **Localidad espacial**: Se refiere a la tendencia de que una vez que se accede a una ubicación específica de memoria, es probable que se acceda a ubicaciones de memoria cercanas en un futuro cercano. Por ejemplo, cuando se lee un bloque de datos de un archivo, es probable que se acceda a los datos que están cerca de ese bloque en lugar de saltar a una ubicación aleatoria en el archivo.

- **Localidad temporal**: Se refiere a la tendencia de que una vez que se accede a un dato específico en la memoria, es probable que se acceda a ese mismo dato nuevamente en un futuro cercano. Por ejemplo, en un bucle que itera sobre un conjunto de datos, es probable que los mismos datos sean accedidos repetidamente.

La forma de acceder a informacion que esta contenida en otra maquina dentro de un cluster va a ser siempre por medio del envio de mensajes.

MULTIPROCESADORES DE MEMORIA COMPARTIDA (Esquema Multiprocesadores)



UMA(acceso a memoria uniforme): Una única memoria compartida, todos los procesadores utilizan la info de esta memoria, y tardan el mismo tiempo en acceder a los recursos que esta contiene.

NUMA(acceso a memoria no uniforme): Cada unidad de procesamiento tiene su cache y su memoria, estos están interconectados entre si por medio de BUSES. Si se requiere la información

que contiene una memoria asociada a la unidad de procesamiento, se obtiene de manera rápida. Mientras que si se necesita un dato que esta en la memoria de otro procesador, sera mas lento, ya que debemos usar el BUS de datos para poder acceder a ella.

Multiprocesadores de memoria Distribuida

Procesadores conectados por una red (local: ej servidores; paralela distribuida como ej: <> maquinas comunicadas por internet), comunicándose por medio de pasaje de mensajes (servidores) . La memoria es local (por lo cual no tenemos problemas de consistencia).

Los cluster estan formadas por distintas maquinas (procesadores multicore), que tienen memoria compartida dentro de cada uno de estas maquinas.

El **Grado de acoplamiento** referido a la cercania y tipo de interconexion entre maquinas, a > grado mas rapidez en cuento a la comunicación, permitiendonos programas de grado fino. Mientras menor sea el grado de acoplamiento, mas lenta la conexión.

>grado de acoplamiento puede ser un rack de servidores. < una red de internet.